Optimizing Pattern Queries for Web Access Logs *

Tadeusz Morzy, Marek Wojciechowski, and Maciej Zakrzewicz

Poznan University of Technology
Institute of Computing Science
ul. Piotrowo 3a, 60-965 Poznan, Poland
Tadeusz.Morzy@put.poznan.pl
Marek.Wojciechowski@cs.put.poznan.pl
Maciej.Zakrzewicz@cs.put.poznan.pl

Abstract. Web access logs, usually stored in relational databases, are commonly used for various data mining and data analysis tasks. The tasks typically consist in searching the web access logs for event sequences that support a given sequential pattern. For large data volumes, this type of searching is extremely time consuming and is not well optimized by traditional indexing techniques. In this paper we present a new index structure to optimize pattern search queries on web access logs. We focus on its physical structure, maintenance and performance issues.

1 Introduction

Web access logs represent the history (the sequences) of users' visits to a web server [11]. Log entries are collected automatically and can be used by administrators for web usage analysis [4][6][7][16][17][18][20]. Usually, after some frequently occurring sequential patterns are discovered [1], the logs are searched for access sequences that contain (support) the discovered sequential patterns. We will refer to this type of searching as to *pattern queries*.

Example web access log is shown in Fig.1. For each client's request we store the client's IP address, the timestamp, and the URL address of the requested object. In general, several requests from the same client may have identical timestamps since they can represent components of a single web page (e.g. attached images). In most cases, web access logs are stored in relational, *SQL*-accessed databases. Let us consider the following example of using the relational approach to pattern queries. Assume that the relation *R(IP,TS,URL)* stores web access sequences. Each tuple contains the sequence identifier (*IP*), the timestamp (*TS*), and the item (*URL*). Our example relation R describes three web access sequences: $\{A,B\} \rightarrow \{C\} \rightarrow \{D\}$, $\{A\} \rightarrow \{E,C\} \rightarrow \{F\}$, and $\{B,C,D\} \rightarrow \{A\}$. Let the searched sequential pattern (subsequence) be: $\{A\} \rightarrow \{E\} \rightarrow \{F\}$. We are looking for all the web access sequences that contain the given sequential

* This work was partially supported by the grant no. KBN 43-1309 from the State Committee for Scientific Research (KBN), Poland.

pattern. Fig.2 gives the relation R and the SQL query, which implements the pattern query.

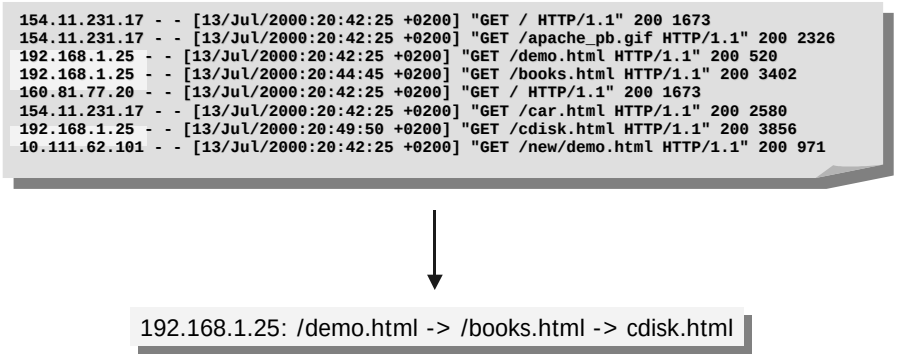

Fig. 1. Web access log example and a web access sequence

IP	TS	URL
1	1	A
1	1	B
1	2	C
1	3	D
2	1	A
2	2	E
2	2	C
2	3	F
3	1	B
3	1	C
3	1	D
3	2	A

```
SELECT IP
FROM   R R1, R R2, R R3
WHERE  R1.IP=R2.IP
  AND  R2.IP=R3.IP
  AND  R1.TS<R2.TS
  AND  R2.TS<R3.TS
  AND  R1.URL='A'
  AND  R2.URL='E'
  AND  R3.URL='F';
```

Fig. 2. The relation of web access sequences and the pattern query

Since web access logs tend to be very large, there is a problem of appropriate optimizing the database access while performing pattern queries, e.g. by means of the above SQL query. Database research has developed many *indexing techniques*, like B^+-trees [5], bitmapped indexes [15], k-d-trees [3], R-trees [10], which are used to optimize queries based on exact matches of single tuples. However, these techniques do not significantly improve pattern queries, which deal with partial matches of multi-tuple sequences. There are also proposals for set-based indexing [8][14], which is used to improve subset searching (e.g. find all papers containing "data mining" and "data warehousing" in a keyword list). However, these methods work for retrieval of unordered sets of items only.

In order to realize the shortcomings of the existing indexing methods, let us consider applying B^+-tree and set-based indexes to execute the query from Fig.2:

1. Using a B^+-tree index, tuples containing all items of each web access sequence are joined first (by IP attribute), and then the verification is done whether they contain the given items in the given order. This approach can be fairly ineffective since a web access sequence may span across many disk block, what results in multiple scanning of each block of the relation.

2. Using a set-based index, the sequence identifiers (*IP* attribute) of all sequences, which contain the searched items in any order, are found, and then the sequences are read from the relation (perhaps with help of a B^+-tree) to verify the ordering of their items. This approach gives much better results, as compared to a B^+-tree index, however, the significant overhead comes from reading and verifying the sequences having incorrect ordering.

In this paper we consider pattern queries on web access log databases. Such databases are characterized by relatively small number of items (URLs), which occur frequently in various order, and therefore a set-based index is not efficient. We present a new bitmap-oriented indexing method, which optimizes the problem of pattern queries. The basic idea behind our method, as compared to set-based indexes, is that the index structure includes not only the items of a sequence, but also the ordering of the items. In this way, we reduce the number of web access sequences needlessly read from the database, what results in shorter query execution time. We performed several experiments, which showed the significant improvement over existing indexing methods.

The structure of the paper is as follows. Section 2 describes the sequential index structure and algorithms to create and to use the index. In Sect.3 we present the results of our performance experiments. Section 4 contains final conclusions.

1.1 Basic Definitions and Problem Formulation

Let $L = l_1, l_2, ..., l_k$ be a set of literals called items (URLs). *Web access sequence* $S =< X_1 X_2 ... X_n >$ is an ordered list of sets of items such that each set of items $X_i \subseteq L$. X_i is called a *sequence element*. All items in a sequence element are unordered. For short, we will also refer to a web access sequence as to a sequence.

We say that a web access sequence $< X_1 X_2 ... X_n >$ is *contained* in another web access sequence $< Y_1 Y_2 ... Y_m >$ if there exist integers $i_1 < i_2 < ... < i_n$ such that $X_1 \subseteq Y_{i_1}, X_2 \subseteq Y_{i_2}, ..., X_n \subseteq Y_{i_n}$.

Problem formulation. Let D be a database of variable length web access sequences. Let S be a web access sequence. The problem of pattern queries consists in finding in D all web access sequences, which contain the web access sequence S.

1.2 Related Work

Database indexes provided today by most database systems are B^+-tree indexes to retrieve tuples of a relation with specified values involving one or more attributes [5]. Each non-leaf node contains entries of the form (v, p) where v is the separator value which is derived from the keys of the tuples and is used to tell which sub-tree holds the searched key, and p is the pointer to its child node. Each leaf node contains entries of the form (k, p), where p is the pointer to the tuple corresponding to the key k.

A set-based bitmap indexing, which is used to enable faster subset search in relational databases was presented in [14] (a special case of superimposed coding). The key idea of the set-based bitmap index is to build binary keys, called *group bitmap keys*, associated with each item set. The group bitmap key represents contents of the item set by setting bits to '1' on positions determined from item values (by means of *modulo* function). An example set-based bitmap index for three item sets: {0, 7, 12, 13}, {2, 4}, and {10, 15, 17} is given in Fig.3. When a subset search query seeking for item sets containing e.g. items 15 and 17 is issued, the group bitmap key for the searched subset is computed (see Fig.4). Then, by means of a bit-wise AND, the index is scanned for keys containing 1's on the same positions. As the result of the first step of the subset search procedure, the item sets identified by set=1 and set=3 are returned. Then, in the verification step (ambiguity of *modulo* function), these item sets are tested for the containment of the items 15 and 17. Finally, the item set identified by set=3 is the result of the subset search. Notice that this indexing method does not consider items ordering.

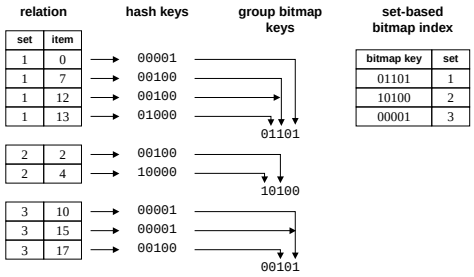

Fig. 3. Set-based bitmap index

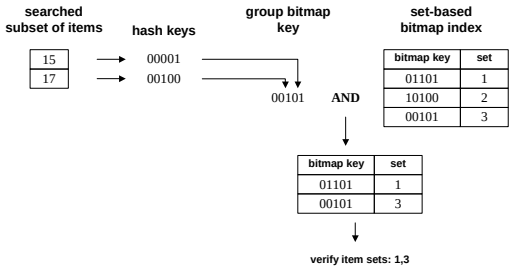

Fig. 4. Set retrieval using set-based bitmap index

In [8], a conceptual clustering method, using *entropic criterion for conceptual clustering* EC^3 is used to define indexing schemes on sets of binary features. Similar data item sets are stored in the same cluster, and similarity measure based on entropy is used during retrieval to find a cluster containing the searched subset. The method does not consider items ordering.

2 Sequential Index Structure

In this section we present our indexing method, called sequential indexing, for optimizing pattern queries. The sequential index structure consists of sequences of bitmaps generated for web access sequences. Each bitmap encodes all items (similarly to a set-based bitmap index) of a portion of a web access sequence as well as ordering relations between each two of the items.

We start with the preliminaries, then we present the index construction algorithm and explain how to use the sequential index structure. Finally, we discuss index storage and maintenance problems.

2.1 Preliminaries

Web access sequences contain categorical items in the form of URLs. For sake of convenience, we convert these items to integer values by means of an *item mapping function*.

Definition 1. *An item mapping function $fi(x)$, where x is a literal, is a function which transforms a literal into an integer value.*

Example 1. Given a set of literals $L = \{A, B, C, D, E, F\}$, an item mapping function can take the following values: $fi(A)$=1, $fi(B)$=2, $fi(C)$=3, $fi(D)$=4, $fi(E)$=5, $fi(F)$=6.

Similarly, we use an *order mapping function* to express web access sequence ordering relations by means of integer values. Thus, we will be able to represent web access sequence items as well as web access sequence ordering uniformly.

Definition 2. *An order mapping function $fo(x, y)$, where x and y are literals and $fo(x, y) \neq fo(y, x)$, is a function which transforms a web access sequence $< \{x\}\{y\} >$ into an integer value.*

Example 2. For the set of literals used in the previous example, an order mapping function can be expressed as: $fo(x, y) = 6 * fi(x) + fi(y)$, e.g. $fo(C, F) = 24$.

Using the above definitions, we will be able to transform web access sequences into item sets, which are easier to manage, search and index. An item set representing a web access sequence is called an *equivalent set*.

Definition 3. *An equivalent set E for a web access sequence $S =< X_1 X_2 ... X_n >$ is defined as:*

$$E = \left(\bigcup_{x \in X_1 \cup X_2 \cup ... \cup X_n} \{fi(x)\} \right) \cup \left(\bigcup_{\substack{x,y \in X_1 \cup X_2 \cup ... \cup X_n : \\ x \ precedes \ y}} \{fo(x, y)\} \right) \quad (1)$$

where: $fi()$ is an item mapping function and $fo()$ is an order mapping function.

Example 3. For the web access sequence $S =< \{A,B\}\{C\}\{D\} >$ and the presented item mapping function and order mapping function, the equivalent set E is evaluated as follows:

$$E = \left(\bigcup_{x\in\{A,B,C,D\}}\{fi(x)\}\right) \cup \left(\bigcup_{\substack{x,y\in\{<\{A\}\{C\}>,<\{B\}\{C\}>,\\ <\{A\}\{D\}>,<\{B\}\{D\}>,<\{C\}\{D\}>\}}} \{fo(x,y)\}\right) =$$

$$= \{fi(A)\}\cup\{fi(B)\}\cup\{fi(C)\}\cup\{fi(D)\}\cup\{fo(A,C)\}\cup\{fo(B,C)\}\cup\{fo(A,D)\}\cup$$
$$\cup\{fo(B,D)\}\cup\{fo(C,D)\} = \{1,2,3,4,9,15,10,16,22\}$$

Observation. For any two web access sequences S_1 and S_2, we have: S_2 contains S_1 if $E_1 \subseteq E_2$, where E_1 is the equivalent set for S_1, and E_2 is the equivalent set for S_2. In general, this property is not reversible.

The size of the equivalent set depends on the number of items in the web access sequence and on the number of ordering relations between the items. For a given number of items in the web access sequence, the equivalent set will be the smallest if there are no ordering relations at all (i.e. $S =< X >$, then $|E| = |X|$, since $E = X$), and will be the largest if S is a sequence of one-item sets (i.e. $S =< X_1X_2...X_n >$, for all i we have $|X_i| = 1$, then $|E| = n + \binom{n}{2}$).

Since the size of an equivalent set quickly increases while increasing the number of the original sequence elements, we split web access sequences into *partitions*, which are small enough to process and encode.

Definition 4. *We say that a web access sequence* $S =< X_1X_2...X_n >$ *is* partitioned *into web access sequences* $S_1 =< X_1...X_{a_1} >$, $S_2 =< X_{a_1+1}...X_{a_2} >$, ..., $S_k =< X_{a_j+1}...X_n >$ *with level* β *if for each web access sequence* S_i *the size of its equivalent set* $|E_i| < \beta$ *and for all* $x, y \in X_1 \cup X_2 \cup ... \cup X_n$, *where* x *precedes* y, *we have: either* $< \{x\}\{y\} >$ *is contained in* S_i *or* $\{x\}$ *is contained in* S_i, *and* $\{y\}$ *is contained in* S_j, *where* $i < j$ *(*β *should be greater than maximal item set size).*

Example 4. Partitioning the web access sequence $S =< \{A,B\}\{C\}\{D\}\{A,F\}\{B\}\{E\} >$ with level 10 results in two web access sequences: $S_1 =< \{A,B\}\{C\}\{D\} >$ and $S_2 =< \{A,F\}\{B\}\{E\} >$, since the sizes of the equivalent sets are respectively: $|E_1| = 9$ ($E_1 = \{1,2,3,4,9,15,10,16,22\}$), and $|E_2| = 9$ ($E_2 = \{1,6,2,5,8,38,11,41,17\}$).

Observation. For a web access sequence S partitioned into S_1, S_2, ..., S_k, and a web access sequence Q, we have: S contains Q if there exists a partitioning of Q into Q_1, Q_2, ..., Q_m, such that Q_1 is contained in S_{i_1}, Q_2 is contained in S_{i_2}, ..., Q_m is contained in S_{i_m}, and $i_1 < i_2 < ... < i_m$.

Our sequential index structure will consist of equivalent sets stored for all web access sequences, optionally partitioned to reduce the complexity. To reduce storage requirements, equivalent sets will be stored in database in the form of *bitmap signatures.*

Definition 5. *The* bitmap signature *of a set X is an N-bit binary number created, by means of bit-wise OR operation, from the hash keys of all data items contained in X. The* hash key *of the item $x \in X$ is an N-bit binary number defined as follows: $hash\_key(X) = 2^{(X \bmod n)}$.*

Example 5. For the set $X = \{0, 7, 12, 13\}$, $N = 5$, the hash keys of the set items are the following:

$hash\_key(0) = 2^{(0 \bmod 5)} = 1 = 00001$,
$hash\_key(7) = 2^{(7 \bmod 5)} = 4 = 00100$,
$hash\_key(12) = 2^{(12 \bmod 5)} = 4 = 00100$,
$hash\_key(13) = 2^{(13 \bmod 5)} = 8 = 01000$.

The bitmap signature of the set X is the bit-wise OR of all items' hash keys:

$bitmap\_signature(X) = 00001\ OR\ 00100\ OR\ 00100\ OR\ 01000 = 01101$.

Observation. For any two sets X and Y, if $X \subseteq Y$ then:

$bitmap\_signature(X)\ AND\ bitmap\_signature(Y) = bitmap_signature(X)$,

where AND is a bit-wise AND operator. This property is not reversible in general (when we find that the above formula evaluates to $TRUE$ we still have to verify the result traditionally).

In order to plan the length N of a bitmap signature for a given average set size, consider the following analysis. Assuming uniform items distribution, the probability that representation of the set X sets k bits to '1' in an N-bit bitmap signature is:

$$P = \frac{\binom{N}{k} f_{k,|X|}}{N^{|X|}}, \text{ where } f_{0,|X|} = 0, f_{q,|X|} = q^{|X|} - \sum_{i=1}^{q-1} \binom{q}{i} f_{i,|X|} \qquad (2)$$

Example probabilistic expected value of number of bits set to '1' for a 16-bit bitmap signatures and various set sizes is illustrated in Fig.5. We can observe that e.g. for a set of 10 items, N should be greater than 8 (else we have all bits set to 1 and the signature is unusable since it is always matched).

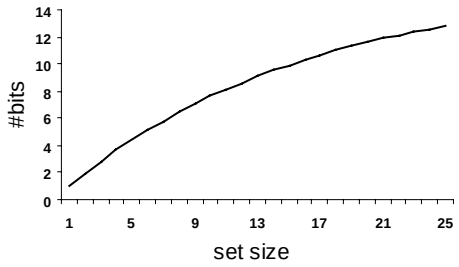

Fig. 5. Number of bitmap signature bits set to '1' for various set sizes (N=16)

The probability that a bitmap signature of the length N having k 1's matches another bitmap signature of the length N having m 1's is $\binom{m}{k}/\binom{N}{k}$. It means

that the smaller k, the better pruning is performed during matching bitmap signatures of item sets, in order to check their containment (so we have to verify less item sets).

2.2 Sequential Index Construction Algorithm

The sequential index construction algorithm iteratively processes all web access sequences in the database. First, the web access sequences are partitioned with the given level β. Then, for each partition of each web access sequence, the equivalent set is evaluated. In the next step, for each equivalent set, its N-bit bitmap signature is generated and stored in the database. The formal description of the algorithm is given below.

Input: database D of web access sequences, partitioning level β, bitmap length N
Output: sequential index for D
Method:

```
for each web access sequence S ∈ D do begin
    partition S into partitions S_1, S_2, ..., S_k with level β;
    for each partition S_i do begin
        evaluate equivalent set E_i for S_i;
        bitmap_i = bitmap_signature(E_i);
        store bitmap_i in the database;
    end;
end.
```

Consider the following example of sequential index construction. Assume that β=10, N=16, and the database D contains three web access sequences: $S_1 = < \{A,B\}\{C\}\{D\}\{A,F\}\{B\}\{E\} >$, $S_2 =< \{A\}\{C,E\}\{F\}\{B\}\{E\}\{A,D\} >$, $S_3 =< \{B,C,D\},\{A\} >$.

First, we partition the web access sequences with β=10. Notice that S_3 is, in fact, not partitioned since its equivalent set is small enough. The symbol $S_{i,j}$ denotes j-th partition of the i-th web access sequence.

$S_{1,1} =< \{A,B\}\{C\}\{D\} >$ (ordering relations are: $A \to C$, $B \to C$, $A \to D$, $B \to D$, $C \to D$)
$S_{1,2} =< \{A,F\}\{B\}\{E\} >$ (ordering relations are: $A \to B$, $F \to B$, $A \to E$, $F \to E$, $B \to E$)
$S_{2,1} =< \{A\}\{C,E\}\{F\} >$ (ordering relations are: $A \to E$, $A \to C$, $E \to F$, $C \to F$)
$S_{2,2} =< \{B\}\{E\}\{A,D\} >$ (ordering relations are: $B \to E$, $B \to A$, $B \to D$, $E \to A$, $E \to D$)
$S_{3,1} =< \{B,C,D\}\{A\} >$ (ordering relations are: $B \to A$, $C \to A$, $D \to A$)

Then we evaluate the equivalent sets for the partitioned web access sequences. We use the example item mapping function and order mapping function taken from the Definitions 1 and 2. The symbol $E_{i,j}$ denotes the equivalent set for $S_{i,j}$.

$E_{1,1} = \{1, 2, 3, 4, 9, 15, 10, 16, 22\}$
$E_{1,2} = \{1, 6, 2, 5, 8, 38, 11, 41, 17\}$
$E_{2,1} = \{1, 3, 5, 6, 11, 9, 36, 24\}$

$E_{2,2} = \{2, 5, 1, 4, 17, 13, 16, 31, 36\}$
$E_{3,1} = \{2, 3, 4, 1, 13, 19, 25\}$

In the next step, we generate 16-bit bitmap signatures for all equivalent sets.

$bitmap\_signature(E_{1,1}) = 1000011001011111$
$bitmap\_signature(E_{1,2}) = 0000101101100110$
$bitmap\_signature(E_{2,1}) = 0000101101111010$
$bitmap\_signature(E_{2,2}) = 1010000000110111$
$bitmap\_signature(E_{3,1}) = 0010001000011110$

Finally, the sequential index is stored in the database in the following form:

SID	bitmap_signature
1	1000011001011111, 0000101101100110
2	0000101101111010, 1010000000110111
3	0010001000011110

2.3 Using Sequential Index for Pattern Queries

During pattern query execution, the bitmap signatures for all web access sequences are scanned. For each web access sequence, the test of a searched subsequence mapping is performed. If the searched subsequence can be successfully mapped to the web access sequence partitions, then the web access sequence is read from the database. Due to the ambiguity of bitmap signature representation, additional verification of the retrieved web access sequence is required. The verification can be performed using the traditional B^+-tree method, since it consists in reading the web access sequence from the database and checking whether it contains the searched subsequence. The formal description of the algorithm is given below. We use a simplified notation of $Q[i\_start..i\_end]$ to denote a partition $< X_{i\_start} X_{i\_start+1} ... X_{i\_end} >$ of a sequence $Q =< X_1 X_2 ... X_n >$, where $1 \leq i\_start \leq i\_end \leq n$. The symbol & denotes bit-wise AND operation.

Input: sequential index, searched subsequence Q
Output: identifiers of web access sequences to be verified
Method:

```
for each sequence identifier sid do begin
    j = 1;
    i_end = 1;
    repeat
        i_start = i_end;
        evaluate equivalence set E_Q for Q[i_start..i_end];
        mask = bitmap_signature(E_Q);
        while mask & bitmap_signature(E_{sid,i}) <> mask
                and j ≤ number of partitions for sid do j++;
        if j ≤ number of partitions for sid then repeat
            i_end++;
            generate equivalence set E_Q for Q[i_start..i_end];
```

```
            mask = bitmap_signature(E_Q);
        until mask & bitmap_signature(E_{sid,i}) <> mask
            or i_end = size of Q;
    until i_start = i_end or j > number of partitions for sid;
    if j ≤ number of partitions then return(sid);
end.
```

Consider the following example of using sequential index to perform pattern queries. Assume that we look for all web access sequences, which contain the subsequence $< \{F\}\{B\}\{D\} >$. We begin with sid=1. We find that $< \{F\} >$ (0000001000000000) matches the first partition (1111101001100001). So, we check whether $< \{F\}, \{B\} >$ (0010001000000000) also matches this partition. Accidentally it does, but when we try $< \{F\}, \{B\}, \{D\} >$ (1010101010000000), we find that it does not match the first partition. Then we move to the second partition to check whether $< \{D\} >$ (0000100000000000) matches the partition (0110011011010000). This test fails and since we have no more partitions, we reject sid=1 (this web access sequence does not contain the given subsequence).

In the next step, we check sid=2. We find that $< \{F\} >$ (0000001000000000) matches the first partition (0101111011010000). So, we check whether $< \{F\}$, $\{B\} >$ (0010001000000000) also matches this partition. It does not, so we move to the second partition and find that $< \{B\} >$ (0010000000000000) matches the partition (1110110000000101). Then we must check whether $< \{B\}, \{D\} >$ (1010100000000000) also matches the partition. This time the check is positive and since we have matched the whole subsequence, we return sid=2 as a part of the result. The web access sequence will be verified later.

Finally, we check sid=3. We find that $< \{F\} >$ (0000001000000000) does not match the first partition (0111100001000100). Since we have no more partitions, we reject sid=3 (this web access sequence does not contain the given subsequence).

So far, the result of our index scanning is the web access sequence identified by sid=2. We still need to read and verify, whether the sequence really contains the searched subset. In our example it does, so the result is returned to the user.

2.4 Physical Storage

Since a sequential index is fully scanned each time a pattern query is performed, it is critical to store it efficiently. We store index entries in the form of $< p, n, bitmap_1, bitmap_2, ..., bitmap_n >$, where p is a pointer to a web access sequence described by the index entry, n is the number of bitmap signatures, and $bitmap_i$ is a single bitmap signature for the web access sequence. The pointer p should address the translation table, which contains pointers to physical tuples of the relation holding the web access sequences (the structure is $< n, p_1, p_2, .., p_n >$). Since we usually have a B^+-tree index on a sequence identifier attribute (to optimize joins), we can use its leaves can as a translation table instead of consuming database space by redundant structures. Example storage implementation for the sequential index from Sect.2.2 is given in Fig.6.

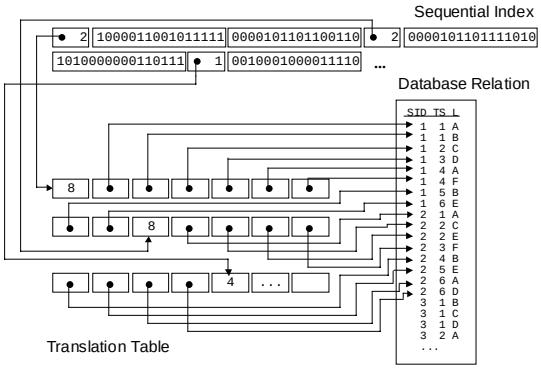

Fig. 6. Example physical storage structure for sequential index

2.5 Update Operations

Maintenance of a sequential index is quite expensive, since bitmap signatures are not reversible, and updates may influence partitioning of web access sequences. For example, when we insert a new tuple into the database, thus extending a web access sequence, we cannot determine what partition should the tuple belong to. Similarly, when we delete a tuple, then both we cannot determine the corresponding partition, and, even if we could do it, we do not know, whether the item being deleted was the only item mapped to a given bit of the bitmap signature (so we could reset the bit).

In order to have a consistent state of a sequential index, we must perform the complete index creation procedure (partitioning, evaluating equivalent sets, generating bitmap signatures) for the web access sequence being modified. However, since this solution might reduce DBMS performance for transaction-intensive databases, we propose the following algorithm of *offline maintenance* for sequential indexes:

1. Whenever a new item is added to an existing web access sequence, we set to '1' all bits in the first bitmap signature for the web access sequence. It means that any subsequence will match the first bitmap signature, and therefore we will not miss the right one. Any false hits will be eliminated during actual verification of subsequence containment.
2. Whenever an item is removed from an existing web access sequence, we do not perform any modifications on the bitmap signatures of the web access sequence. We may get false hits, but they will be eliminated during final verification.

Notice that using the above algorithm, the overall index performance may decrease temporarily, but we will not get incorrect query results. Over a period of time, the index should be rebuild either completely, or for updated web access sequences only, e.g. according to a transaction log.

3 Experimental Results

We have performed several experiments on synthetic data sets to evaluate our sequential indexing method. The database of web access sequences was generated randomly, with uniform item distribution, and stored by *Oracle8* DBMS. We used dense data sets, i.e. the number of available items was relatively small, and therefore each item occurred in a large number of web access sequences. The web access sequences contained 1-item sets only (pessimistic approach - maximal number of ordering relations).

Figure 7a shows the number of disk blocks (including index scanning and relation access), which were read in order to retrieve web access sequences containing subsequences of various lengths. The data set contained 50000 web access sequences, having 20 items of 50 in average. The compared database accessing methods were: traditional *SQL* query using B^+-tree index on *IP* attribute (B^+-tree), 24-bit set-based bitmap index (24S), 32-bit sequential index with $\beta = 28$ built on top of 24-bit set-based bitmap index (24S32Q28), and 48-bit sequential index with $\beta = 55$ built on top of 24-bit set-based bitmap index (24S48Q55). Our sequential index achieved a significant improvement for the searched subsequences of length greater than 4, e.g. for the subsequence length of 5 we were over 20 times faster than the B^+-tree method and 8 times faster than the set-based bitmap index.

We also analyzed the influence of the partitioning level β value on the sequential index performance. Figure 7b illustrates the filtering factor (percentage of web access sequences matched) for three sequential indexes built on bitmap signatures of total size of 48 bits, but with different partitioning. We noticed that partitioning web access sequences into a large number of partitions (small β) results in performance increase for long subsequences, but worsens the performance for short subsequences. Using a small number of web access sequence partitions (high β) results in more "stable" performance, but the performance is worse for long subsequences.

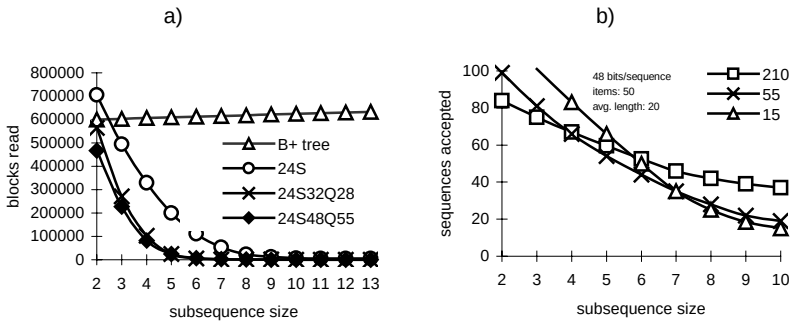

Fig. 7. Experimental results

4 Final Conclusions

Pattern queries on web access logs are specific in the sense that they require complicated *SQL* queries and database access methods (multiple joins, inefficient optimization). In this paper we have presented the new indexing method, called sequential indexing, which can replace a B^+-tree indexing and set-based indexing. During experiments, we have found that the most efficient solution is to combine a set-based index (which checks items of a web access sequence) with a sequential index (which checks the items ordering), what results in dramatic outperforming B^+-tree access methods.

References

1. Agrawal R., Srikant R.: Mining Sequential Patterns. Proc. of the 11th ICDE Conf. (1995)
2. Bayardo R.J.: Efficiently Mining Long Patterns from Databases. Proc. of the ACM SIGMOD International Conf. on Management of Data (1998)
3. Bentley J.L.: Multidimensional binary search trees used for associative searching. Comm. of the ACM 18 (1975)
4. Catledge L.D., Pitkow J.E.: Characterizing Browsing Strategies in the World Wide Web. Proc. of the 3rd Int'l WWW Conference (1995)
5. Comer D.: The Ubiquitous B-tree. Comput. Surv. 11 (1979)
6. Cooley R., Mobasher B., Srivastava J.: Data preparation for mining World Wide Web browsing patterns. Journal of Knowledge and Information Systems 1 (1999)
7. Cooley R., Mobasher B., Srivastava J.: Grouping Web Page References into Transactions for Mining World Wide Web Browsing Patterns. Proc. of the 1997 IEEE Knowledge and Data Engineering Exchange Workshop (1997)
8. Diamantini C., Panti M.: A Conceptual Indexing Method for Content-Based Retrieval. Proc. of the 15th IEEE Int'l Conf. on Data Engineering (1999)
9. Guralnik V., Wijesekera D., Srivastava J.: Pattern Directed Mining of Sequence Data. Proc. of the 4th KDD Conference (1998)
10. Guttman A.: R-trees: A dynamic index structure for spatial searching. Proc. of ACM SIGMOD International Conf. on Management of Data (1984)
11. Luotonen A.: The common log file format. http://www.w3.org/pub/WWW/ (1995)
12. Mannila H., Toivonen H.: Discovering generalized episodes using minimal occurrences. Proc. of the 2nd KDD Conference (1996)
13. Mannila H., Toivonen H., Verkamo A.I.: Discovering frequent episodes in sequences. Proc. of the 1st KDD Conference (1995)
14. Morzy T., Zakrzewicz M.: Group Bitmap Index: A Structure for Association Rules Retrieval. Proc. of the 4th KDD Conference (1998)
15. O'Neil P.: Model 204 Architecture and Performance. Proc. of the 2nd International Workshop on High Performance Transactions Systems (1987)
16. Perkowitz M., Etzioni O.: Adaptive Web Sites: an AI challenge. Proc. of the 15th Int. Joint Conf. AI (1997)
17. Pirolli P., Pitkow J., Rao R.: Silk From a Sow's Ear: Extracting Usable Structure from the World Wide Web. Proc. of Conf. on Human Factors in Computing Systems (1996)

18. Pitkow J.: In search of reliable usage data on the www. Proc. of the 6th Int'l WWW Conference (1997)
19. Srikant R., Agrawal R.: Mining Sequential Patterns: Generalizations and Performance Improvements. Proc. of the 5th EDBT Conference (1996)
20. Yan T.W., Jacobsen M., Garcia-Molina H., Dayal U.: From User Access Patterns to Dynamic Hypertext Linking. Proc. of the 5th Int'l WWW Conference (1996)